解读Cardinality Estimation算法（第三部分：LogLog Counting）

作者 张洋 | 发布于 2013-01-03

算法 基数估计 大数据 LogLogCounting

上一篇文章介绍的Linear Counting算法相较于直接映射bitmap的方法能大大节省内存（大约只需后者1/10的内存），但毕竟只是一个常系数级的降低，空间复杂度仍然为$O(N_{max})$。而本文要介绍的LogLog Counting却只有$O(log_2(log_2(N_{max})))$。例如，假设基数的上限为1亿，原始bitmap方法需要12.5M内存，而LogLog Counting只需不到1K内存（640字节）就可以在标准误差不超过4%的精度下对基数进行估计，效果可谓十分惊人。

本文将介绍LogLog Counting。

简介

LogLog Counting（以下简称LLC）出自论文“Loglog Counting of Large Cardinalities”。LLC的空间复杂度仅有$O(log_2(log_2(N_{max})))$，使得通过KB级内存估计数亿级别的基数成为可能，因此目前在处理大数据的基数计算问题时，所采用算法基本为LLC或其几个变种。下面来具体看一下这个算法。

基本算法

均匀随机化

与LC一样，在使用LLC之前需要选取一个哈希函数H应用于所有元素，然后对哈希值进行基数估计。H必须满足如下条件（定性的）：

1、H的结果具有很好的均匀性，也就是说无论原始集合元素的值分布如何，其哈希结果的值几乎服从均匀分布（完全服从均匀分布是不可能的，D. Knuth已经证明不可能通过一个哈希函数将一组不服从均匀分布的数据映射为绝对均匀分布，但是很多哈希函数可以生成几乎服从均匀分布的结果，这里我们忽略这种理论上的差异，认为哈希结果就是服从均匀分布）。

2、H的碰撞几乎可以忽略不计。也就是说我们认为对于不同的原始值，其哈希结果相同的概率非常小以至于可以忽略不计。

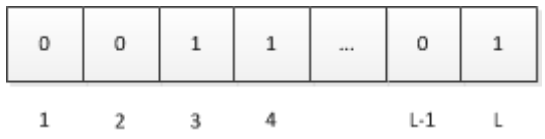
3、H的哈希结果是固定长度的。

以上对哈希函数的要求是随机化和后续概率分析的基础。后面的分析均认为是针对哈希后的均匀分布数据进行。

思想来源

下面非正式的从直观角度描述LLC算法的思想来源。

设a为待估集合（哈希后）中的一个元素，由上面对H的定义可知，a可以看做一个长度固定的比特串（也就是a的二进制表示），设H哈希后的结果长度为L比特，我们将这L个比特位从左到右分别编号为1、2、…、L：



又因为a是从服从均与分布的样本空间中随机抽取的一个样本，因此a每个比特位服从如下分布且相互独立。

$$P(x=k)=\begin{cases}0.5(k=0)\\0.5(k=1)\end{cases}$$

通俗说就是a的每个比特位为0和1的概率各为0.5，且相互之间是独立的。

设$\rho(a)$为a的比特串中第一个“1”出现的位置，显然$1 \le \rho(a) \le L$，这里我们忽略比特串全为0的情况（概率为$1/2^L$）。如果我们遍历集合中所有元素的比特串，取ρ_{max}为所有$\rho(a)$的最大值。

此时我们可以将$2^{\rho_{max}}$作为基数的一个粗糙估计，即：

$$\hat{n}=2^{\rho_{max}}$$

下面解释为什么可以这样估计。注意如下事实：

由于比特串每个比特都独立且服从0-1分布，因此从左到右扫描上述某个比特串寻找第一个“1”的过程从统计学角度看是一个伯努利过程，例如，可以等价看作不断投掷一个硬币（每次投掷正反面概率皆为0.5），直到得到一个正面的过程。在一次这样的过程中，投掷一次就得到正面的概率为$1/2$，投掷两次得到正面的概率是$1/2^2$，…，投掷k次才得到第一个正面的概率为$1/2^k$。

现在考虑如下两个问题：

1、进行n次伯努利过程，所有投掷次数都不大于k的概率是多少？

2、进行n次伯努利过程，至少有一次投掷次数等于k的概率是多少？

首先看第一个问题，在一次伯努利过程中，投掷次数大于k的概率为$1/2^k$，即连续掷出k个反面的概率。因此，在一次过程中投掷次数不大于k的概率为$1-1/2^k$。因此，n次伯努利过程投掷次数均不大于k的概率为：

$$P_n(X \leq k) = (1-1/2^k)^n$$

显然第二个问题的答案是：

$$P_n(X \geq k) = 1-(1-1/2^{k-1})^n$$

从以上分析可以看出，当$n \ll 2^k$时，$P_n(X \geq k)$的概率几乎为0，同时，当$n \gg 2^k$时，$P_n(X \leq k)$的概率也几乎为0。用自然语言概括上述结论就是：当伯努利过程次数远远小于2^k时，至少有一次过程投掷次数等于k的概率几乎为0；当伯努利过程次数远远大于2^k时，没有一次过程投掷次数大于k的概率也几乎为0。

如果将上面描述做一个对应：一次伯努利过程对应一个元素的比特串，反面对应0，正面对应1，投掷次数k对应第一个“1”出现的位置，我们就得到了下面结论：

设一个集合的基数为n，ρ_{max}为所有元素中首个“1”的位置最大的那个元素的“1”的位置，如果n远远小于$2^{\rho_{max}}$，则我们得到ρ_{max}为当前值的概率几乎为0（它应该更小），同样的，如果n远远大于$2^{\rho_{max}}$，则我们得到ρ_{max}为当前值的概率也几乎为0（它应该更大），因此$2^{\rho_{max}}$可以作为基数n的一个粗糙估计。

分桶平均

上述分析给出了LLC的基本思想，不过如果直接使用上面的单一估计量进行基数估计会由于偶然性而存在较大误差。因此，LLC采用了分桶平均的思想来消减误差。具体来说，就是将哈希空间平均分成m份，每份称之为一个桶（bucket）。对于每一个元素，其哈希值的前k比特作为桶编号，其中$2^k = m$，而后L-k个比特作为真正用于基数估计的比特串。桶编号相同的元素被分配到同一个桶，在进行基数估计时，首先计算每个桶内元素最大的第一个“1”的位置，设为M[i]，然后对这m个值取平均后再进行估计，即：

$$\hat{n} = 2^{\frac{1}{m}\sum M[i]}$$

这相当于物理试验中经常使用的多次试验取平均的做法，可以有效消减因偶然性带来的误差。

下面举一个例子说明分桶平均怎么做。

假设H的哈希长度为16bit，分桶数m定为32。设一个元素哈希值的比特串为“0001001010001010”，由于m为32，因此前5个bit为桶编号，所以这个元素应该归入“00010”即2号桶（桶编号从0开始，最大编号为m-1），而剩下部分是“01010001010”且显然$\rho(01010001010) = 2$，所以桶编号为“00010”的元素最大的ρ即为M[2]的值。

偏差修正

上述经过分桶平均后的估计量看似已经很不错了，不过通过数学分析可以知道这并不是基数n的无偏估计。因此需要修正成无偏估计。这部分的具体数学分析在“Loglog Counting of Large Cardinalities”中，过程过于艰涩这里不再具体详述，有兴趣的朋友可以参考原论文。这里只简要提一下分析框架：

首先上文已经得出：

$$P_n(X \leq k) = (1-1/2^k)^n$$

因此：

$$P_n(X = k) = (1-1/2^k)^n - (1-1/2^{k-1})^n$$

这是一个未知通项公式的递推数列，研究这种问题的常用方法是使用生成函数（generating function）。通过运用指数生成函数和poissonization得到上述估计量的Poisson期望和方差为：

$$\varepsilon_n \sim [(\Gamma(-1/m)\frac{1-2^{1/m}}{log2})^m + \epsilon_n]n$$

$\nu_n \sim [(\Gamma(-2/m)\frac{1-2^{2/m}}{log2})^m - (\Gamma(-1/m)\frac{1-2^{1/m}}{log2})^{2m} + \eta_n]n^2$

其中$|\epsilon_n|$和$|\eta_n|$不超过10^{-6}。

最后通过depoissonization得到一个渐进无偏估计量：

$\hat{n} = \alpha_m 2^{\frac{1}{m}\sum M[i]}$

其中：

$\alpha_m = (\Gamma(-1/m)\frac{1-2^{1/m}}{log2})^{-m}$

$\Gamma(s) = \frac{1}{s}\int_0^\infty e^{-t}t^s dt$

其中m是分桶数。这就是LLC最终使用的估计量。

误差分析

不加证明给出如下结论：

$E_n(\hat{n})/n = 1 + \theta_{1,n} + o(1)$

$\sqrt{Var_n(E)}/n = \beta_m/\sqrt{m} + \theta_{2,n} + o(1)$

其中$|\theta_{1,n}|$和$|\theta_{2,n}|$不超过10^{-6}。

当m不太小（不小于64）时，β大约为1.30。因此：

$StdError(\hat{n}/n) \approx \frac{1.30}{\sqrt{m}}$

算法应用

误差控制

在应用LLC时，主要需要考虑的是分桶数m，而这个m主要取决于误差。根据上面的误差分析，如果要将误差控制在ϵ之内，则：

$m > (\frac{1.30}{\epsilon})^2$

内存使用分析

内存使用与m的大小及哈希值得长度（或说基数上限）有关。假设H的值为32bit，由于$\rho_{max} \leq 32$，因此每个桶需要5bit空间存储这个桶的ρ_{max}，m个桶就是$5 \times m/8$字节。例如基数上限为一亿（约2^{27}），当分桶数m为1024时，每个桶的基数上限约为$2^{27}/2^{10} = 2^{17}$，而$log_2(log_2(2^{17})) = 4.09$，因此每个桶需要5bit，需要字节数就是$5 \times 1024/8 = 640$，误差为$1.30/\sqrt{1024} = 0.040625$，也就是约为4%。

合并

与LC不同，LLC的合并是以桶为单位而不是bit为单位，由于LLC只需记录桶的ρ_{max}，因此合并时取相同桶编号数值最大者为合并后此桶的数值即可。

小结

本文主要介绍了LogLog Counting算法，相比LC其最大的优势就是内存使用极少。不过LLC也有自己的问题，就是当n不是特别大时，其估计误差过大，因此目前实际使用的基数估计算法都是基于LLC改进的算法，这些改进算法通过一定手段抑制原始LLC在n较小时偏差过大的问题。后面要介绍的HyperLogLog Counting和Adaptive Counting就是这类改进算法。